TESTING VALIDATION TOOLS ON CLIPS-BASED EXPERT SYSTEMS

C.L. Chang, R.A. Stachowitz and J.B. Combs

Lockheed Missiles & Space Company, Inc.
Software Technology Center, B/254E, O/96-10
3251 Hanover Street
Palo Alto, Ca 94304

Abstract. The Expert Systems Validation Associate (EVA) is a validation system which was developed at the Lockheed Software Technology Center and Artificial Intelligence Center between 1986 and 1990. EVA is an integrated set of generic tools to validate any knowledge-based system written in any expert system shell such as CLIPS, ART, OPS5, KEE and others. Many validation tools have been built in the EVA system. In this paper, we describe the testing results of applying the EVA validation tools to the Manned Maneuvering Unit (MMU) Fault Diagnosis, Isolation, and Reconfiguration (FDIR) expert system, written in C Language Integrated Production System (CLIPS), obtained from the NASA Johnson Space Center.

INTRODUCTION

The Expert Systems Validation Associate (EVA) is a validation system developed at the Lockheed Software Technology Center and Artificial Intelligence Center. Its goal was to develop generic and flexible tools to validate any knowledge-based system written in any expert system shell such as CLIPS, ART, OPS5, KEE and others. We achieved this goal by using the *metaknowledge approach.* Metaknowledge, or knowledge about knowledge, describes constraints on the knowledge that can be used for checking redundancy, completeness, consistency, and correctness of a knowledge base. We realized that we can not standardize validation criteria for all applications. Each application has its own statements to be validated, but they can be represented by means of the metapredicates.

From 1986 to 1990, we implemented many different checkers (Stachowitz et al. 1987,1991, Chang et al. 1988,1989,1990, McGuire et al. 1990) for checking different types of errors in a knowledge base. In order to see if the checkers were sufficient to detect anomalies and errors, we tested them on a real expert system, the Manned Maneuvering Unit (MMU) Fault Diagnosis, Isolation, and Reconfiguration (FDIR) expert system, written in C Language Integrated Production System (CLIPS), we obtained from the NASA Johnson Space Center. To validate the FDIR expert system, we implemented a CLIPS translator to convert FDIR's CLIPS rules into the internal knowledge representations in Prolog used by our checkers. EVA only requires a new translator for a new shell because its checkers were built to be independent of any particular expert system shell. This paper describes the testing results of applying the EVA validation tools to the FDIR expert system.

MMU FDIR AUTOMATION TASK

The MMU FDIR automation task is described in (Lawler and Williams 1988). The primary objectives of the task were to investigate the potential of automating the fault diagnosis, isolation and reconfiguration (FDIR) function of the MMU currently performed by the MMU pilot.

The problem of this task is to determine the types of all possible failure scenarios, appropriate instrumentation for monitoring the MMU state and supporting diagnosis, and appropriate actuators for executing the failure recovery.

The MMU system is represented by a three-level architectural diagram. At level 0, MMU is a black box whose inputs and outputs are:

Inputs of MMU:
- MMU Torques
- Automatic Attitude Hold (AAH) Select
- Gyro Power
- Control-Electronic-Assembly-A (CEA-A) Power
- Control-Electronic-Assembly-B (CEA-B) Power
- CEA-A Isolate
- CEA-B Isolate
- Translation Commands
- Rotation Commands
- Xfeed Valves

Outputs of MMU:
- Tank-A Pressure (Gas Flow Rate)
- Tank-B Pressure (Gas Flow Rate)
- Thruster-A Firings
- Thruster-B Firings

At level 1, MMU is decomposed into 3 components, namely, Gyros, CEA Assembly (CEA-A and CEA-B), and Tank/Thruster Assembly (A and B Thrusters). Their inputs and outputs are:

Inputs of Gyros:
- MMU Torques
- Gyro Power

Outputs of Gyros:
- Gyro Rotation Commands

Inputs of CEA Assembly:
- Automatic Attitude Hold (AAH) Select
- Control-Electronic-Assembly-A (CEA-A) Power
- Control-Electronic-Assembly-B (CEA-B) Power
- Translation Commands
- Rotation Commands
- Gyro Rotation Commands

Outputs of CEA Assembly:
- Valve-Drive-Amplifier-A Commands
- Valve-Drive-Amplifier-B Commands

Inputs of Tank/Thruster Assembly:
- CEA-A Isolate
- CEA-B Isolate
- Xfeed Valves
- Valve-Drive-Amplifier-A Commands
- Valve-Drive-Amplifier-B Commands

Outputs of Tank/Thruster Assembly:
- Tank-A Pressure (Gas Flow Rate)
- Tank-B Pressure (Gas Flow Rate)

Thruster-A Firings
Thruster-B Firings

At level 2, the CEA Assembly is further decomposed into CEA-A and CEA-B each of which knows about the power status of the other. Similarly, the Tank/Thruster Assembly is broken down into the components of side A and side B coupled through the crossfeed valves.

The architectural diagram provides a simple description of the causal effect of MMU. To implement the recovery procedures, MMU provides one or more of the recovery options: do nothing and continue mission, turn on/off CEA-A or CEA-B, turn on/off gyro power, turn on/off AAH, open/close crossfeed valves, or abort mission (return to orbiter or call for rescue).

Measurements and switch settings of different parts/components in MMU are taken by instruments to assess the state of the MMU system and components. Each measurement is analyzed for the amount of diagnostic capacity it contributes to the FDIR system. The measurements and switch settings are processed and converted into symbolic information represented by a set of initial facts in CLIPS. The initial facts are processed by CLIPS rules in the FDIR expert system to generate diagnosis and recovery actions.

CLIPS TRANSLATOR

We implemented a CLIPS translator to convert FDIR's CLIPS rules into the internal Prolog representations used by the EVA checkers. The CLIPS translator needs to handle many special CLIPS constructs such as multifield variables, nested AND/OR, variable assignments, if...then...else, while...do. Since Prolog does not support these constructs, the CLIPS translator needs to perform normalizations and semantic translations. Therefore, it is possible that a CLIPS rule may be translated into many Prolog Horn clauses. For example, the CLIPS rule:

```
(defrule pull-block-c
    ?s <- (stack $?stack-1 c $?stack-2)
    =>
    (assert (stack $?stack-1 $?stack-2))
    (retract $s))
```

where $?stack-1 and $?stack-2 are multifield variables is translated into

```
rule('pull-block-c',
    [stack(X),
     append(Stack_1, [c | Stack_2], X),
     append(Stack_1, Stack_2, Z)],
    [assert(stack(Z)),
     retract(stack(X))]).
rule('append-0',
    [ ],
    [append([ ],L,L)]).
rule('append-1',
    [append(T,L,R)],
    [append([H | T],L,[H | R])]).
```

VALIDATION OF THE FDIR EXPERT SYSTEM

The FDIR expert system consists of 104 CLIPS rules. The CLIPS translator converted these CLIPS rules into 357 normalized internal rules which were then checked by the EVA checkers. The goal of our experiments was to detect if redundancy, incompleteness, inconsistency, or incorrectness is present in FDIR's CLIPS rules. The following were the results we found:

Redundancy

There are indeed redundancies in FDIR's CLIPS rules. The redundancy checker detected redundancies in the following 5 CLIPS rules:

```
;1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;improper CEA behavior
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;logic for (no aah) or (no gyro movement)and(aah on) - prime mode
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule cea-test-input-null
        (or (aah off) (and (gyro on)(gyro movement none none)))
        (rhc roll none pitch none yaw none)
        (thc x none y none z none)
        (vda ?side ?thrust on)
=>
        (assert (failure cea))
        (printout crlf "failure - vda signal was sent to " crlf)
        (printout "thrusters without intended command "crlf)
)


;2
;pos roll gyro input
(defrule cea-a-gyro-input-roll-pos-6
       (aah off) (gyro on)
       (gyro movement roll pos)
        (side a on)
        (side b on)
        (rhc roll none pitch none yaw none)
        (thc x none y none z none)
        (vda a ?m on)

=>
        (assert (failure cea))
        (assert (suspect a))
        (printout crlf "aah failed to correct pos roll")
)


;3
(defrule cea-b-gyro-input-roll-pos-6
       (aah off) (gyro on)
       (gyro movement roll pos)
        (side a on)
        (side b on)
        (rhc roll none pitch none yaw none)
        (thc x none y none z none)
        (or
        (vda b r4 off)
        (vda b l1 off)
        (vda b ?n&~r4&~l1 on)
        )
```

```
=>
        (assert (failure cea))
        (assert (suspect b))
        (printout crlf "aah failed to correct pos roll")
)


;4
;pos roll gyro input
(defrule gyro-input-roll-pos-backup-a-6
        (aah off) (gyro on)
        (gyro movement roll pos)
         (not (checking thrusters))
         (side a on)
         (side b off)
         (rhc roll none pitch none yaw none)
         (thc x none y none z none)
         (or
         (vda a r4 off)
         (vda a l1 off)
         (vda a ?n&~r4&~l1 on)
         )
=>
        (assert (failure cea))
        (assert (suspect a))
        (printout crlf "cea failure on side a")
)


;5
(defrule gyro-input-roll-pos-backup-b-6
        (aah off) (gyro on)
        (gyro movement roll pos)
         (not (checking thrusters))
         (side a off)
         (side b on)
         (rhc roll none pitch none yaw none)
         (thc x none y none z none)
         (or
         (vda b r4 off)
         (vda b l1 off)
         (vda b ?n&~r4&~l1 on)
         )
=>
        (assert (failure cea))
        (assert (suspect b))
        (printout crlf "cea failure on side b")
)
```

where the last 4 rules contain parts *subsumed* by the first rule. For example, in the second rule, *cea-a-gyro-input-roll-pos-6*, the conditions

```
         (aah off)
         (rhc roll none pitch none yaw none)
         (thc x none y none z none)
```

(vda a ?m on)

are subsumed by the left hand side (LHS) of the first rule, and both rules have the same action, (assert (failure cea)), in the right hand side (RHS). This type of redundancy, which is not obvious to the naked eye, can be detected only when the rules are normalized.

Incompleteness

An input to the FDIR expert system is a set of initial facts. The requirements that can be used for checking incompleteness should describe what are all the legal inputs. That is, we want to verify if every input can be processed by the FDIR expert system. Unfortunately, these requirements are not specified in (Lawler and Williams 1988). We could only manually deduce from the 6 test cases given in Lawler and Williams' report and from the MMU model given in the previous section that an input contains one or more instances of each of the following CLIPS relations:

```
(defrelation aah
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type WORD) (allowed-words on off)))

(defrelation fuel-used-a
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type NUMBER) (range 0 ?VARIABLE)))

(defrelation fuel-used-b
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type NUMBER) (range 0 ?VARIABLE)))

(defrelation side
   (min-number-of-fields 3)
   (max-number-of-fields 3)
   (field 2 (type WORD) (allowed-words a b))
   (field 3 (type WORD) (allowed-words on off)))

(defrelation xfeed-a
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type WORD) (allowed-words open closed)))

(defrelation xfeed-b
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type WORD) (allowed-words open closed)))

(defrelation tank-pressure-current
   (min-number-of-fields 3)
   (max-number-of-fields 3)
   (field 2 (type WORD) (allowed-words a b ab))
   (field 3 (type NUMBER) (range 0 ?VARIABLE)))

(defrelation tank-pressure-was
   (min-number-of-fields 3)
```

```
   (max-number-of-fields 3)
   (field 2 (type WORD) (allowed-words a b ab))
   (field 3 (type NUMBER) (range 0 ?VARIABLE)))

(defrelation gyro
   (min-number-of-fields 2)
   (max-number-of-fields 2)
   (field 2 (type WORD) (allowed-words on off)))

(defrelation gyro
   (min-number-of-fields 4)
   (max-number-of-fields 4)
   (field 2 (type WORD) (allowed-words movement))
   (field 3 (type WORD) (allowed-words none pitch roll yaw))
   (field 4 (type WORD) (allowed-words none pos neg)))

(defrelation vda
   (min-number-of-fields 4)
   (max-number-of-fields 4)
   (field 2 (type WORD) (allowed-words a b))
   (field 3 (type WORD) (allowed-words b1 b2 b3 b4
                                       f1 f2 f3 f4
                                       l1 l2 l3 l4
                                       r1 r2 r3 r4
                                       u1 u2 u3 u4
                                       d1 d2 d3 d4))
   (field 4 (type WORD) (allowed-words on off)))

(defrelation rhc
   (min-number-of-fields 7)
   (max-number-of-fields 7)
   (field 2 (type WORD) (allowed-words roll))
   (field 3 (type WORD) (allowed-words none pos neg))
   (field 4 (type WORD) (allowed-words pitch))
   (field 5 (type WORD) (allowed-words none pos neg))
   (field 6 (type WORD) (allowed-words yaw))
   (field 7 (type WORD) (allowed-words none pos neg)))

(defrelation thc
   (min-number-of-fields 7)
   (max-number-of-fields 7)
   (field 2 (type WORD) (allowed-words x))
   (field 3 (type WORD) (allowed-words none pos neg))
   (field 4 (type WORD) (allowed-words y))
   (field 5 (type WORD) (allowed-words none pos neg))
   (field 6 (type WORD) (allowed-words z))
   (field 7 (type WORD) (allowed-words none pos neg)))
```

The types of incompleteness we checked were deadend and unreachable literals. A deadend literal is a LHS literal of a rule that does not match with all input facts or all RHS literals in the knowledge base. Obviously, any rule containing a deadend literal will not fire. An unreachable literal is a RHS literal of a rule that never appears in the LHS of a rule. In our experiment, we did not find deadend literals. However, we did find 2 unreachable literals, (assert (failure-thrusters-with-xfeed)), and (assert (checking thruster)) in the following rule:

```
(defrule xfeed-fuel-reading-test-general
        (declare (salience -10))
        ?x <- (xfeed-a open)
        ?y <- (xfeed-b open)
        (fuel-used-a ?fuel-a)
        (fuel-used-b ?fuel-b)
        (tank-pressure-was ab ?p-old)
        (tank-pressure-current ab ?p-new)
        (test (!= (- ?p-old (+ ?fuel-a ?fuel-b)) ?p-new))
        (side b on)
        (side a on)
=>
        (retract ?x ?y)
        (assert (xfeed-a closed))
        (assert (xfeed-b closed))
        (assert (failure-thrusters-with-xfeed))
        (printout crlf "failure occurred while executing thruster commands")
        (printout crlf crlf "xfeed is open, testing sides after closing xfeed")
        (printout crlf crlf)
        (assert (checking thruster))
)
```

It turned out that (assert (checking thruster)) is a typo. The correct one should be (assert (checking thrusters)).

Inconsistency

In the EVA system, we provide the flexibility of letting the user define his own inconsistency criteria for his expert system. This is done through specifications of constraints on input facts and asserted facts. We introduced some new constructs such as *defambiguity*, *defincompatibility* and *defimply* for this purpose. For example, if we state

```
(defambiguity (assert ?fact) (retract ?fact))
```

then an ambiguity will occur when there is a set of input facts that can lead to a situation where a rule may fire to assert ?fact and another rule may fire to retract ?fact. If we state

```
(defincompatibility (xfeed-a closed) (xfeed-a open))
```

then we have an inconsistency when both facts, (xfeed-a closed) and (xfeed-a open), simultaneously exist in the short term memory. If we state

```
(defimply
    (gyro movement ~none ?v)
=>
    (!= ?v none))
```

then we have to make sure whenever a fact about gyro is asserted, if its third field is not 'none' then its fourth field must not be 'none'. Adhering to EVA standards, we use the same syntax of CLIPS in this case for constraint specifications so that the user does not have to learn a new syntax.

Based upon some of the above constraint specifications, our logic checker detected anomalies where one rule added a fact, followed by the deletion of the same fact by another rule, and in between these two operations no new operations were performed, as specified by the above *defambiguity* statement.

Incorrectness

There are two major methods, namely, verification and testing (Chang et al. 1990), for check-

ing correctness of an expert system. Both these methods assume that requirements on functional behaviors of the system exist. Behavior requirements specify input-output relationships of the system. Unfortunately, such requirements are not specified in (Lawler and Williams 1988). The approach given in their report is to perform testing of the FDIR expert system with a few manually created test cases, and then manually evaluate the output produced by the system.

The EVA system has a prototype tool called test case generator which automatically generates test cases that satisfy the input specifications given in the previous sections. However, we are not in a position to tell what the correct outputs would be for the generated test cases. The performance for them has to be evaluated by experts in the field.

CONCLUDING REMARKS

We have described the testing results of the validation experiments we performed for the FDIR expert system. The experiments are very useful. They not only tell us that EVA is a useful validation system to detect errors/anomalies in a knowledge base, but also reveal to us what requirements and information EVA needs in order to perform thorough validation of an expert system. Because of these experiments, we gained better insight on how to build an expert system development environment that can nicely integrate requirement specification, system prototyping/coding, and system verification, validation and testing.

REFERENCES

Chang, C.L., and Stachowitz, R.A. [1988] "Testing expert systems," *Proc. Space Operations Automation and Robotics (SOAR 88) Workshop*, Wright State University, Dayton, OH.

Chang, C.L., Stachowitz, R.A., and Combs, J.B. [1989] "Testing integrated knowledge-based systems," *Proc. of IEEE International Workshop on Tools for Artificial Intelligence*, 1989, IEEE Computer Society Press.

Chang, C.L., Combs, J.B., and Stachowitz, R.A. [1990] "A report on the expert systems validation associate (EVA)," *Journal of Expert Systems With Applications*, Vol. 1, pp.217-230, 1990, Pergamon Press, Maxwell House, Fairview Park, Elmsford, New York 10523.

Chang, C.L., Stachowitz, R.A., and Combs, J.B. [1990] "Validation of nonmonotonic knowledge-based systems," *Proc. of the IEEE Second International Conference on Tools for Artificial Intelligence*, 1990, IEEE Computer Society Press.

Lawler, D.G., and Williams, L.J.F. [1988] *MMU FDIR Automation Task --- Final Report*, Contract NAS9-17650, Task Order EC87044, McDonnell Douglas Astronautics Company - Engineering Services, 16055 Space Center Blvd., Houston, TX 77062.

McGuire, J. [1990] "Uncovering redundancy and rule-inconsistency in knowledge bases via deduction," *Proc. of Fifth Annual Conference on Computer Assurance: Systems Integrity and Software Safety (IEEE COMPASS-90)*, June, 1990, IEEE Computer Society Press.

McGuire, J., and Stiles, R. [1990] "Detecting interference in knowledge-based systems," *Proc. of the 5th Knowledge-Based Software Assistant (KBSA) Conference*, Rome Laboratory, Griffiss Air Force Base, New York 13441-5700.

Stachowitz, R.A., and Combs, J.B. [1987] "Validation of expert systems," *Proc. of the Twentieth Hawaii International Conference on System Sciences*, Kona, Hawaii, January,

1987.

Stachowitz, R.A., Combs, J.B., and Chang, C.L. [1987] "Validation of knowledge-based systems," *Proc. AIAA/NASA/USAF Symp. on Automation, Robotics and Advanced Computing*, Arlington, VA, Mar 1987; published also as: Chapter 7 in *Machine Intelligence for Aerospace Systems*, eds. H. Lum and E. Heer, *Progress Series in Astronautics and Aeronautics*, Vol. 101, ed. Martin Summerfield, Am. Institute of Astronautics and Aeronautics, 1989.

Stachowitz, R.A., and Chang, C.L. [1991] "Completeness checking of knowledge-based systems," *Proc. 24th Hawaii International Conference on System Sciences*, Kona, HI, Jan 1991.